Frankenstein Cross Assemblers Users Manual

Mark Zenier

1. Disclaimer

This software is in the public domain. Any prior copyright claims are relinquished.

This software is distributed with no warranty whatever. The author takes no responsibility for the consequences of its use.

2. Introduction

2.1 This Manual

This manual covers the basic features for the Frankenstein cross assemblers. The specific features for a given target architecture will be covered in the appendix for that cross-assembler.

2.2 Notation

Items enclosed in [] are optional. The "[]" are not entered in the source statement or command line, and are just a notation.

3. Invocation

name [-o binary] [-l listing] [-s symbols] [-d] input

The optional operands on the command line can be in any order.

3.1 Input File

The input file must appear after the optional entries on the command line. Only one input file is used with this version. If the file name is a single minus sign, the standard input is used.

3.2 Listing File

    -l filename

An annotated listing will only be produced if an optional list file is provided, otherwise the error messages and summary will be output to the console. There is no required suffix or default extension for the file name.

3.3 Hex Output File

    -h filename or
    -o filename

The binary output will only be produced if the optional file is provided. See the section in the description of the output files for the format of the data records. If any errors occur, the hex output file is either not created or is deleted depending on whether the error occurred in the input or output phases of processing. There is no required suffix or default extension for the file name.

3.4 Symbol Listing File

    -s filename

The symbol table is dumped with one entry per line for whatever use is desired, normally with the sort utility to produce symbol listings in a different order that the one provided. See the section on the output files for a description of the format. There is no required suffix or default extension for the file name.

3.5 Debug

The -d debug option

- Saves the intermediate file in the /usr/tmp directory
- Calls the abort() system call to produce a "core" file

3.6 Processor Selection

assembler name or
-p string

Some of these assemblers support related families of processors, and can be limited to specific subsets of the total instructions by sending the processor number or name as an operand. The operand is scanned for unique substrings which identify the target processor. The operand can be either the name of the executable, or the operand of the -p option, with the operand of the -p having precedence. The name of the executable may not available in some operating systems. The CPU pseudo-operation has precedence over both the name scan and the -p operand [see the appendix for the specific processor].

Example

a6800 for the mc6800 instruction set
a6801 for the 6801/6803 instruction set
a6811 for the mc68hc11 instruction set

4. Source Input File

The source input is a text file a with one statement per line. Adjacent symbols must be separated by spaces or tabs or special characters such as operators or parenthesis.

4.1 Input Lines

Source lines consist of up to four parts

[Label] [Opcode [Operands]] [; comments]

Labels — A symbol that starts in the first column of the line is a label. A label is used as the defining instance of a symbol, the place where it is given a value, normally the address of the location in memory where the data generated by the rest of the statement is placed. Labels are optional for target machine instructions statements. Label are required on some pseudo-operations, and are not allowed on others.

Opcode — The first symbol in a line that doesn't start in column 1 is treated as an opcode, all symbols after that are treated as symbol references. Opcodes are treated as a separate set of symbols, so a label can have the same character string as an opcode. Opcodes are converted to uppercase for comparison, so case is not significant.

Operands — The rest of the line is the operands associated with the opcode. These are expressions, symbol references, or other syntactic elements determined by the specific operation.

Comments — Comments start with a ';', anything on a line after a semicolon is ignored. Lines that consist of only a comment are treated as empty lines.

Example

```
; just a little example
        org $700
label   equ *  ; set label to current location
        adda    #$21 ; a 6800 example
        end
```

4.1.1 Limitations Input lines should be shorter than 256 characters. An error message is issued if lines are longer.

4.1.2 Empty Empty lines are ignored as input, but are copied to the listing file.

4.2 Syntactic Elements

4.2.1 Symbols Symbols are strings of characters. The first of which is one of the following.

A-Z a-z ! & ˆ _ ˜

The rest of the string consists of zero or more of the following.

0-9 A-Z a-z ! & ˆ _ ˜

There is no length limit to symbols, all characters are significant. Case is significant.

4.2.2 Reserved Variables Names Machine specific names, and operator names that can appear in symbolic expressions share the same symbol table with labels, and cannot be used as labels. The list will vary from machine to machine. Case is significant for these symbols.

Examples

and defined eq ge gt high le low lt mod ne not or shl shr xor AND DEFINED EQ GE GT HIGH LE LOW LT MOD NE NOT OR SHL SHR XOR

4.2.3 Numeric Constants numeric constants can be specified in decimal, hexadecimal, octal, and binary. Constants are maintained as long variables, but the instruction generation will cause an error if the value is too big to fit in its destination field.

4.2.3.1 Decimal Constants

— one or more decimal characters (0-9) followed by an optional "d" or "D" base designator.

4.2.3.2 Hexadecimal Constants

Either

— one or more hexadecimal characters (0-9 a-f A-F) preceded by a "$" base designator.

— a decimal character (0-9) followed by zero or more hexadecimal characters (0-9 a-f A-F) followed by a "h" or "H" base designator

4.2.3.3 Octal Constants

Either

— one or more octal characters (0-7) preceded by a "@" base designator

— one or more octal characters (0-7) followed by a "o" or "O" or "q" or "Q" base designator. Note: this is the letter "O" not the number "0".

4.2.3.4 Binary Constants

Either

— one or more binary characters (0 or 1) preceded by a "%" base designator.

— one or more binary characters (0 or 1) followed by a "b" or "B" base designator.

4.2.3.5 Examples

123	decimal number
123d	decimal number
$faf	hexadecimal number
0fafh	hexadecimal number, note leading zero used to differentiate this from symbol "fafh"

@1234	octal number
1234Q	octal number
%010101	binary number
010101b	binary number

4.2.4 String Constants String constants are specified using either the quotation mark " or the apostrophe '. A string starting with one of these character is terminated by only that character and can contain the other string delimiter as data. A string with no characters is valid.

The values generated in the binary output can be different from the host computers character set (the default). See the section on Defining Target Character Sets.

Within the character string, the backslash is an escape character.

\c Character escapes

> The only characters defined for the default (no) translation are \\ \" \´. Any other translation, (the control characters '\n', '\r', etc.) require a translation table be defined. Values can be set up for any character following the backslash escape except x and 0 through 7.

\777 Octal character escapes

> An one, two or three character octal constant. The one byte value is the binary representation of the number. The value is masked off for a maximum value of 255.

\xFF Hexadecimal character escapes

> A one or two character hexadecimal constant, preceded by a lower case x. Upper and lower case characters A through F are valid. The one byte value is the binary representation of the number.

Examples

```
"this isn't illegal"
'this is the other delimiter'
'"'
"\xfe\0\n"
```

4.3 Expressions

Expressions consist of

— Symbolic References
— Location Counter References
— Numeric Constants
— String Constants
— Operators

Expressions are used as operands in statements where any numeric value or memory address is desired.

4.3.1 Symbols A symbol reference is the name of an item in the symbol table, which has a numeric value associated with it. This value is either the memory address of the statement which has this symbol as a label, or the value of the expression defined in a SET or EQU statement which has this symbol as a label.

4.3.1.1 Forward References During the input phase of processing, a symbol value may not be known if the definition of the symbol has not yet occurred. Some pseudo-operations require that their operand expressions have a value at the input phase, so no references to as yet undefined symbols can occur in this case. In the output phase of processing, it will result in an error if any of the symbols used in expressions do not have values defined.

4.3.1.2 Reserved Symbols for Operators Using reserved symbols as labels will result in a syntax error as they are predefined as a different type of syntactic element than the rest of the symbol table. Operators for which there is no special character representation, and items like machine register names and condition code types will be represented by reserved symbols. The set of the reserved symbols will vary for each target machine, and will be documented in the appendix for each target.

4.3.2 Location Counter The special name '*' used in an expression represents the location of the first byte of the binary output for the current statement. Some assemblers use some other character for source code compatiblity.

4.3.3 Constants The form of numeric constants is defined in the previous section.

4.3.4 Strings String constants, as defined in the previous section, are valid in expressions. However, at most the first two characters are used. If the string is the null string, i.e. "", the numeric value is zero. If the string is one character long, the value is the value of the current character set translation, or the host computers character set if no translation is active. If the string is two or more characters long, the value is 256*(the first character) + (the second character).

4.3.5 Operators

4.3.5.1 Description

'+' expression

Unary plus. No real effect.

'-' expression

Unary minus, the result is the negative of the expression.

NOT expression

Logical negation. The result is 0 if the expression is nonzero, 1 if the expression is 0.

HIGH expression

Result is the High half of a two byte number, equivalent to (expression SHR 8) AND $FF.

LOW expression

Result is the Low order byte of the expression, equivalent to (expression AND $FF)

expression '*' expression
expression '/' expression
expression '+' expression
expression '-' expression

Standard arithmetic operations.

expression MOD expression

Result is the remainder of the division of the value of the left expression by the right.

expression SHL expression
expression SHR expression

Shift the value of the left expression left or right by the number of bit positions given by the right operation.

expression GT expression

expression GE expression
expression LT expression
expression LE expression
expression NE expression
expression EQ expression
expression '>' expression
expression '>=' expression
expression '<' expression
expression '<=' expression
expression '=' expression
expression '<>' expression

Relational expressions. If the relation is true, the value is 1. If false, the value is 0. The operators are nonassociative, the expression " 1 < 3 < 5 " is not legal.

expression AND expression
expression OR expression
expression XOR expression

Bitwise logical operations.

DEFINED symbol

If the symbol (not an expression) is defined, i.e. used as a label, before this point in the input the value is 1. If not, the value is 0.

'(' expression ')'

Parenthesis are available to override the operator precedence.

4.3.5.2 Precedence The precedence of the operators from lowest to highest.

1. HIGH LOW
2. OR XOR
3. AND
4. NOT
5. GT GE LE LT NE EQ '>' '>=' '<' '<=' '=' '<>'
6. '+' '-'
7. '*' '/' MOD SHL SHR
8. unary '+' '-'
9. (expression)

5. Statements

The names for the operations and pseudo-operations for each assembler are specified in the adaptation files, and can be different from the examples given here.

5.1 Label Only Line

A line with only a label starting in column 1 will define that symbol with the current value of the location counter.

5.2 End

[Label] END

The End statement terminates the processing of the current file. For an include file, the file is closed and input resumes in the file that contained the include statement. For the main file, processing shifts to the following passes of the assembly. The end statement is optional, as the end of file condition is treated in the same fashion. If the optional label is present, The symbol specified is used as the execution start address that is output in the binary file. The symbol must be used as a label somewhere else in the file. When more than one start address is specified, the last one in the file is used.

5.3 File Inclusion

INCLUDE "filename"
INCL "filename"

The include statement shifts input from the current file to the file specified. Input resumes from the file containing the include statement when the end of file or the End statement is reached in the included file. Includes can be nested up to the limits of the include file stack, currently a limit of 20 deep, or the limits of the operating system, whichever comes first. Includes can be recursive, i.e., a file can include itself. If a file cannot be opened, either do to an bad filename or a lack of system resources, an error is issued.

5.4 Conditional Assembly

5.4.1 If

IF expression

The IF statement allows selective assembly. If the expression evaluates to a nonzero value, all statements between the IF and the matching ELSE or ENDI are assembled. If the expression evaluates to zero, or the expression in noncomputable due to a forward reference, all statements between the IF and the matching ELSE or ENDI are ignored.

Note: it is safer to use the DEFINED operator when testing for the existence of a symbol than to rely on the noncomputability of an expression.

IF statements can be nested to a depth determined by a configuration constant, currently 32. No label is allowed on an IF statement.

5.4.2 Else

ELSE

The ELSE statement causes all statements between it and its corresponding ENDI statement to be treated the opposite of the statements between the matching IF and this statement. When the expression on the matching IF is nonzero, the statements between the ELSE and ENDI are ignored. If the IF expression failed, the statements between the ELSE and the ENDI are assembled. Labels are not allowed on ELSE statements.

5.4.3 End If

ENDI

The ENDI statement terminates processing of its matching IF statement. Labels are not allowed on ENDI statements.

5.5 Symbolic Constants

Symbols can be assigned numeric values with the SET and EQU statements. The expressions cannot have forward references to as yet undefined symbols.

5.5.1 Equate

Label EQU expression

The EQU statement takes the value of the expression and creates a symbol with that value. Symbols defined in EQU statements cannot already exist, or be redefined.

5.5.2 Set

Label SET expr

The SET statement sets the symbol specified in the label field with the numeric value of the expression. The SET statement can change the value of a symbol, but only if the symbol is originally defined in a previous SET statement.

Example

```
counter set 1
counter set counter+1
counter set counter+1
```

5.6 Location Counter Value Setting

The address of the generated binary data can be changed with the ORG and reserve statements.

5.6.1 Org

```
[Label] ORG expression
```

The location counter is set to the numeric value of expression. It is an error if the expression contains references to symbols not yet defined. The optional label is set to the new value of the location counter.

5.6.2 Reserve Memory

```
[Label] RMB expression
[Label] RESERVE expression
```

The reserve memory statement moves the location counter forward by the number of bytes specified in the expression. The label is set to the first location of this area.

5.7 Data Definitions

5.7.1 Define Byte Data

```
[Label]  BYTE  expression [, expression] ...
[Label]  FCB   expression [, expression] ...
[Label]  DB  expression [, expression] ...
```

The define byte statement generates one character of data for each expression in the expression list. There can be up to 128 expressions on a line, more than the line length will allow. The optional label is set to the first location of this area.

5.7.2 Define Word Data

```
[Label]  WORD  expression [, expression] ...
[Label]  FDB   expression [, expression] ...
[Label]  DW  expression [, expression] ...
```

The define word statement generates a two byte integer for each expression in the expression list. There can be up to 128 expressions on a line, more than the line length will allow. The byte order of the data is determined by the adaptation files for the target processor. The optional label is set to the first location of this area.

5.7.3 Define String Data

```
[Label] STRING string [, string] ...
[Label] FCC string [, string] ...
```

The define string statement generates data encoded in the current character set translation, one byte per character, excluding the delimiter characters. The optional label is set to the first location of this area.

5.8 Defining Target Character Sets

The values generated for String Constants in both the Define String Data and in expressions can be specified on a character by character basis. This is to support cross assembly where the target system has a different character set from the host computer.

5.8.1 Define Character Set Translation

Label CHARSET

The define character set translation statement defines a name and creates an internal table for a character set. The label symbol is treated like the label on an EQU statement. The value is from an internal counter and has little or no meaning outside of using it on a CHARUSE statement to specify which translation to use. There can be up to 5 [configurable] character translation sets. A CHARSET statement must precede any CHARDEF statements.

5.8.2 Define Character Value

CHARDEF string, expression [, expression] ...
CHD string, expression [, expression] ...

The define character value statement set the translation for one or more characters in the table defined by the preceding CHARSET statement. There can be more than one character in the string, but the number of expression in the value list must match the number of characters. Octal and Hexadecimal escape sequences cannot occur in the string.

There are two sets in each translation table. The first is for the characters, the second for characters escaped with the backslash.

Note: the characters ’"’ and ’\"’ (and "’" and "\’") each have an entry in different halves of the translation tables.

5.8.3 Use Character Translation

CHARUSE
CHARUSE expression

The use character translation statement changes the translation for the following statements. The statement without an expression turns off the translation, so the host character set is used. The statement with an expression (the name given on the CHARSET statement) sets the translation to the set defined in the respective CHARSET.

Example

```
ascii   charset
        chardef " !\"#$%&\’",$20,$21,$22,$23,$24,$25,$26,$27
        chardef "()*+,-./",$28,$29,$2a,$2b,$2c,$2d,$2e,$2f
        chardef "01234567",$30,$31,$32,$33,$34,$35,$36,$37
        chardef "89:;<=>?",$38,$39,$3a,$3b,$3c,$3d,$3e,$3f
        chardef "@ABCDEFG",$40,$41,$42,$43,$44,$45,$46,$47
        chardef "HIJKLMNO",$48,$49,$4a,$4b,$4c,$4d,$4e,$4f
        chardef "PQRSTUVW",$50,$51,$52,$53,$54,$55,$56,$57
        chardef "XYZ[\\]^_",$58,$59,$5a,$5b,$5c,$5d,$5e,$5f
        chardef "‘abcdefg",$60,$61,$62,$63,$64,$65,$66,$67
        chardef "hijklmno",$68,$69,$6a,$6b,$6c,$6d,$6e,$6f
        chardef "pqrstuvw",$70,$71,$72,$73,$74,$75,$76,$77
        chardef "xyz{|}~",$78,$79,$7a,$7b,$7c,$7d,$7e
        chardef ’"’, $22 ; not the same table entry as ’\"’
        chardef "’", $27 ;
        chardef "\n\t\v\b\r\f\a", $0a,$09,$0b,$08,$0d,$0c,$07
        charuse ascii
```

5.9 Machine Instructions

[Label] opcode operands?

Machine instructions generate the binary output by evaluating the expressions for the operands, and matching the opcode with the entry in the instruction generation tables.

If the instruction has more than one format which is selected by the value of the operands, the selection criteria must able to be determined at the input phase of processing. For example, in the mc6800 architecture direct address mode, any memory variables that are have an address between 0 and 255 must be defined before any reference to these symbols.

The optional label is set to the first location of the generated instruction.

6. Output

6.1 Program Generated Messages and Errors

6.1.1 Messages

“ ERROR SUMMARY - ERRORS DETECTED {count}”
“ - WARNINGS {count}”

output at the end of the listing and on the console.

6.1.2 System Errors

“cannot open hex output {filename}”

file cannot be opened for output. The assembly continues as if the -[oh] option was not specified.

“cannot open input file {filename}”

file cannot be opened for reading. Fatal error.

“cannot open list file {filename}”

file cannot be opened for output. The assembly continues as if the -l option was not specified.

“cannot open symbol file {filename}”

file cannot be opened for output. The assembly continues as if the -s option was not specified.

“cannot open temp file {filename}”

file cannot be opened for input or output. Fatal error.

“no input file”

no input operand specified. Fatal error.

“no match on CPU type {string}, default used”

operand for a -p option can’t be matched.

“cannot allocate string storage”

Request to operating system for more memory failed. Fatal error. The string storage pool is where the character representation of symbols and names for include files are stored.

“cannot allocate symbol space”

Request to operating system for more memory failed. Fatal error. The symbol table is full. The symbol space is the set of arrays where the symbol values and other numeric information is stored.

“cannot redefine reserved symbol”

Error in defining reserved symbols. Two calls to "reservedsym()" with the same character string value exist in the setup. Fatal error. Should not occur in a production executable.

“excessive number of subexpressions”

The first pass ran out of element storage for the expression parse tree. The expression is too complex. Internal error which should never occur. Fatal error.

''unable to allocate symbol index''
: Request to operating system for more memory failed. Fatal error. The symbol table is full. The symbol index is used in the output pass to direct symbol references to the symbol table entry.

6.1.3 Error

''ELSE with no matching if''
''ENDI with no matching if''
: mismatched if/else/endi results in an else/endi left over

''IF stack overflow''
: more than IFSTKDEPTH (32) nested if statements

''Overlength/Unterminated Line''
: line longer than input buffer, or not terminated with newline character.

''active IF at end of file''
: mismatched if/else/endi results in an unclosed if

''cannot change symbol value with EQU''
: symbol is already defined.

''cannot change symbol value with SET''
: symbol is already defined, but not with a Set statement.

''cannot create character translation table''
: the internal table for a character translation set cannot be allocated due to lack of space, or more translation sets than the assembler is configured for.

''cannot open include file''
: The include file cannot be opened for reading, or not enough system resources are available to open the file.

''character already defined 'char' ''
: the character is already present in a previous CHARDEF statement for this translation set.

''expression exceeds available field width''
: The value of an expression is too large to fit into the field of the instruction. Relative branch target is too far away.

''expression fails validity test''
: An explicit test programmed in the generation string for the instruction failed. These conditions are documented in the appendix for the specific instruction set.

''expression stack overflow''
: Too many level of parenthesis or complex expression with operator precedence that results in the expression evaluation stack overflowing.

''include file nesting limit exceeded''
: include files are nested to more than FILESTKDPTH (20) levels deep. "cannot open include file" usually occurs first.

''invalid char in instruction generation''
: Internal error, instruction generation string is not defined properly. Should not occur in a production executable.

“invalid character constant ’char’ ”

A character specification in a string constant isn’t properly formed.

“invalid character to define ’char’ ”

a constant in the string in a CHARDEF statement is of a form (octal, hex, or improperly formed) that does not have a translation table entry.

“invalid opcode”

No such string occurs in the opcode symbol table. Opcode strings are converted to uppercase before comparison, and therefore are case insensitive.

“invalid operands”
“invalid operands/illegal instruction for cpu”

statement has a valid opcode, with the correct syntax, but no code generation can be found in the table for these operands

“invalid syntax for instruction”

opcode is valid, but not for this syntax form

“more characters than expressions”
“more expressions than characters”

A mismatch between the string constant and the number of expressions in a CHARDEF statement.

“multiple definition of label”

label symbol is already defined.

“no CHARSET statement active”

a CHARDEF statement occurs before any CHARSET statement.

“noncomputable expression for EQU”
“noncomputable expression for ORG”
“noncomputable expression for SET”
“noncomputable result for RMB expression”

expression contains reference to symbols that have not yet been defined, and thus has no numeric value.

“nonexistent character translation table”

expression in a CHARUSE statement does not correspond to any CHARSET statements label.

“overflow in instruction generation”
“overflow in polish expression conversion”

The intermediate file line being built exceeds the length of the buffer. The expression is too complex.

“syntax error at/before character ^{character}”
“syntax error at/before character {character}”
“syntax error at/before token {symbol/constant}”
“syntax error at invalid token {constant/string} ”
“syntax error at/before string {string} ”
“syntax error at/before End of Line”
“syntax error at/before {relational op}”
“syntax error at/before Undeterminable Symbol”

“syntax error - undetermined yyerror type”
: statement is in a form that the first pass parser cannot recognize. The next syntactic element is inappropriate for whatever language element the parser is working on.

“error or premature end of intermediate file”
“syntax error - cannot backup”
“unimplemented width”
“unknown intermediate file command”
“yacc stack overflow”
: Internal errors, should not occur.

“undefined character value ’char’ ”
: A string constant contains a character not defined in a CHARDEF statement for the current character translation.

“undefined symbol {symbolname}”
: symbol has no definition anywhere in file.

6.1.4 Warnings

“character translation value truncated”
: An expression in a CHARDEF statment has a value less than zero or greater then 255.

“forward reference to SET/EQU symbol”
: A symbol in an expression is defined in a set/equ statement that occurs after the line. For set statements, the value of the symbol is that defined in the set statement closest to the end of the file.

“string constant in expression more than 2 characters long”
: The first two characters are used as the numeric value of the subexpression.

6.2 Listing

When the -l option is used, the detailed listing is output to the given file. This consists of the symbol listing followed by the annotated listing.

6.2.1 Symbol Table The symbol listing is printed three symbols across, with the value then name of the symbol. Undefined symbols will have "????????" in their value field. The symbols are listed in order of first occurrence in the input. Only the first fifteen characters of a symbol are printed.

6.2.2 Instruction Lines The source statements are printed in the same form as they were input with no reformatting. Following the source line, will be any error or warning messages associated with the line. Statements which generate data will be preceded with the address and data for them in hexadecimal format. If more than six bytes of data are generated, the remainder will be printed on the following lines, with up to sixteen bytes per line. All data generated is printed in the file. Statements that don’t generate data but have some value oriented operation, like EQU, SET, ORG, or RESERVE, will print that value in the first 24 columns of the source line.

6.3 Symbol File

When the -s option is used, the symbol table is printed to the given file. The format is one symbol per line, address then symbol name. If the symbol is undefined, "????????" is printed for the address. The symbols are printed in the order of first occurrence, either definition or reference, in the source file. This feature is provided so the system sort utility can be used to produce symbol tables sorted by either address or name. The entire symbol name is printed.

6.4 Binary Output

6.4.1 Intel Hex Record Format The Intel hex record is a printable text string with an ASCII character representing 4 bits of a byte. The characters used are "0" through "9" and "a" through "f", representing binary data 0000 to 01001, and 1010 through 1111. There are always two ASCII characters used to represent 1 byte, the high half, then the low half. There is one record per line in a text file. This format is accepted by most of prom programmers.

Record Layout

:ccaaaattddddddddddkk

':'	indicator character
cc	ascii-hex representation of an 8 bit count of data characters
aaaa	ascii-hex representation of 16 bit address
tt	ascii-hex representation of an 8 bit type
	the address is the first location to store the data
	the address is the location to start execution, the count is 00, no data is present.
dd...	ascii-hex representation of the data, two characters per byte. The number of 8 bit bytes is given by the count field.
kk	ascii-hex representation of an 8 bit checksum. The sum of the binary representation, modulo 256, of the count, the high half of the address, the low half of the address, the type, all the data bytes, and the checksum must be zero.

7. Program Limits

Line Length	256 characters including end of line control characters
Number of Symbols	
	— The maximum allocatable block size divided by the size of a pointer. For a iAPX286, about 16000 entries.
	— The amount of allocatable memory divided by the amount of space per symbol. For an iAPX286, 22 + 1 + the length of the string bytes.
Symbol Length	See line length
Output	A 16 bit address in the output record format limits output to 65536 binary bytes.
Expressions per Line	128 (in BYTE and WORD statements)
Strings per Line	128 (in STRING statements)
Nested If Statements	32 levels
Nested Include Files	20 or whatever the operating system allows
Subexpressions per Line	258 symbols, constants, operators (total)
Character Translation Sets	5 sets, plus the default (host) character set

CONTENTS